AVL Trees
AVL Tree Definition (§9.2)

AVL trees are balanced.

An AVL Tree is a binary search tree such that for every internal node $v$ of $T$, the heights of the children of $v$ can differ by at most 1.

An example of an AVL tree where the heights are shown next to the nodes.
### Height of an AVL Tree

**Fact:** The *height* of an AVL tree storing $n$ keys is $O(\log n)$.

**Proof:** Let us bound $n(h)$: the minimum number of internal nodes of an AVL tree of height $h$.

- We easily see that $n(1) = 1$ and $n(2) = 2$.
- For $n > 2$, an AVL tree of height $h$ contains the root node, one AVL subtree of height $n-1$ and another of height $n-2$.
- That is, $n(h) = 1 + n(h-1) + n(h-2)$

Knowing $n(h-1) > n(h-2)$, we get $n(h) > 2n(h-2)$. So

$n(h) > 2n(h-2)$, $n(h) > 4n(h-4)$, $n(h) > 8n(h-6)$, … (by induction),

$n(h) > 2^in(h-2i)$

$n(h) > 2^{\frac{h}{2} - 1}$

Taking logarithms: $h < 2\log n(h) + 2$

Thus the height of an AVL tree is $O(\log n)$
Insertion in an AVL Tree

- Insertion is as in a binary search tree.
- Always done by expanding an external node.
- Example:

```
before insertion

44
    /   \
  17     78
 /       /   \
32      50     88
 /     /     /   \
48    62    48    62

after insertion

44
    /   \
  17     78
 /       /   \
32      50     88
 /     /     /   \
48    62    54
```

\[\text{a=y, b=x, c=z} \]
unbalanced...

...balanced
let \((a,b,c)\) be an inorder listing of \(x, y, z\) perform the rotations needed to make \(b\) the topmost node of the three

- \(a=z\)
- \(b=y\)
- \(c=x\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

- \(T_0\)
- \(T_1\)
- \(T_2\)
- \(T_3\)

\(a=z\)
\(c=x\)
\(b=y\)

\(a=z\)
\(c=y\)
\(b=x\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

\(T_0\)
\(T_1\)
\(T_2\)
\(T_3\)

- case 1: single rotation (a left rotation about \(a\))
- case 2: double rotation (a right rotation about \(c\), then a left rotation about \(a\))

(other two cases are symmetrical)
Removal in an AVL Tree

Removal begins as in a binary search tree, which means the node removed will become an empty external node. Its parent, \( w \), may cause an imbalance.

Example:

```
before deletion of 32
```

```
after deletion
```
Let $z$ be the first unbalanced node encountered while traveling up the tree from $w$. Also, let $y$ be the child of $z$ with the larger height, and let $x$ be the child of $y$ with the larger height. (!tie!)

We perform $\text{restructure}(x)$ to restore balance at $z$.

As this restructuring may upset the balance of another node higher in the tree, we must continue checking for balance until the root of $T$ is reached.
Running Times for AVL Trees

- a single restructure is $O(1)$
  - using a linked-structure binary tree
- find is $O(\log n)$
  - height of tree is $O(\log n)$, no restructures needed
- insert is $O(\log n)$
  - initial find is $O(\log n)$
  - Restructuring up the tree, maintaining heights is $O(\log n)$
- remove is $O(\log n)$
  - initial find is $O(\log n)$
  - Restructuring up the tree, maintaining heights is $O(\log n)$